

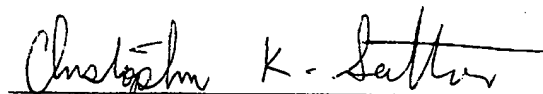


Docket No. 10090531-1
USPTO Ser. No. 09/992,224

Affidavit Under 37 C.F.R. §1.131

1. I, Christopher K. Sutton, am an inventor of the claimed subject matter of US Patent application No. 09/992,224 Agilent attorney docket no. 10990531-1.
2. The invention claimed in the referenced patent application was conceived and reduced to practice in the USA.
3. I understand that willful false statements and the like contained herein are punishable by fine, imprisonment, or both and may jeopardize the validity of the application or any patent issuing thereon.
4. The invention claimed in the referenced patent application was reduced to practice as a software program on a date prior to October 23, 2001. A reduction to practice is evidenced by the attached "Exec 3 Windows NT Test Executive" Programmer's Guide prepared after the actual reduction to practice of the invention claimed in the reference U.S. Patent Application. The attached document is a portion of the Programmer's Guide written to help employees of Agilent use the reduction to practice of the invention as part of an internal assessment of the usability and utility of the software in a manufacturing and production environment.

I declare that all statements made herein are based upon my own knowledge and to the best of my knowledge are true and all statements made on information and belief are believed to be true.


Christopher K. Sutton

Exec 3

Windows NT Test Executive

Programmer's Guide

- † Test Software
- † Plug-In Extensions
- † User Interface Extensions
- † Administration



Agilent Technologies

Lake Stevens Site

<http://>

[xec3](#)

Agilent Restricted

Table of Contents

Exec 3 Windows NT Test Executive	1
Table of Contents	3
Exec 3 Overview	5
Purpose	5
Overview	6
Design Philosophy	7
Definitions	8
User Interface	9
Test Results Files	11
Writing Test Software	12
Overview	12
Writing Procedure Definition Code	17
Writing Test Code	20
Writing Other TestSw.cls Code	24
Writing TestSystem and Dut Code	26
Using Wizards to Help Write Code	29
Tips and Advanced Topics	31
Helper Applications	33
Exec 3 Details	34
Document and Code Conventions	34
Procedure Objects	37
ProcRun and Result Objects	45
Dut and Test System Objects	48
Interface Objects: TestSw, Plugin, Exec, UserMenu	54
State Machine	63
ActiveX Interface API	65
Exec 3 Plug-Ins	67
COM/ActiveX Details	68
Test Results Files	70
Administering Exec 3	73
Installing Exec 3	73
Configuring Exec 3	73

Exec 3 Overview

Purpose

The purpose of the Exec3 program is to efficiently test products. Here, "efficiently test" is evaluated over many phases of a product life cycle: test development, qualification testing, and manufacturing test. "Products" includes complex products with that require many parametric measurements.

To aid in test development and test re-use, Exec3 provides a standardized software interfaces for the test developer. Exec3 relieves the test developer of the tasks presenting an operator interface, sequencing tests, displaying and logging results, and interfacing with other manufacturing systems.

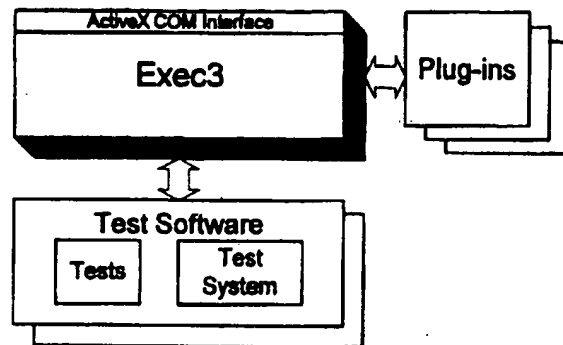
During qualification testing such as Immunity and Environmental, Exec3 provides a consistent programmatic interface for qualification test supervisor programs.

During manufacturing, Exec3 provides a common operator interface for multiple products, and helps maximize test productivity. It also provides a consistent software interface to other manufacturing systems.

Overview

Exec3 is a 32-bit Windows NT application. Its graphical user interface allows the user to select a specific product model to be tested, enter configuration information, select and edit a test procedure, start testing and control test execution (pause, continue, repeat...), and view the test results.

Exec3 has well-defined COM (Component Object Model) interfaces to *test software* and *Exec3 plug-ins*. Test software is the code components provided by a test developer to test a specific product. This includes tests, test procedures, and test system drivers. Exec3 plug-ins are code components that allow Exec3 to be interfaced to other systems, such as a database or equipment calibration verification system. Test software and plug-in code is contained in DLL (Dynamic Link Library) files, so they can be developed and delivered independently.



In addition to the test software and plug-in interfaces, Exec3 exposes an ActiveX COM interface for controlling it. This allows testing to be initiated and monitored from other programs.

Exec3 test software is most easily developed using Visual Basic. Exec3's COM interface includes an object model that aids in test and procedure organization. It encourages *parameter-driven* test routines, which aids in re-use.

Design Philosophy

Design Objectives:

Exec3 design choices were made with these goals in mind..

- Run on readily available computer platforms that supports interfaces to our future products (Intel-based PCs, not Series 300)
- Run on a supported, standard, preferred, easily networked, and powerful operating system (Windows NT, not RMB)
- Allow test development in a language that is not obsolete, and is the most productive language for most engineers (Visual Basic, not RMB)
- Continue to use and evolve the useful features of Exec II. These features include parameter-driven test code, the procedure/sequence file, a hierarchy of procedure/test/datapoint, line/customer/marginal specification limits, dynamic loading of test software.
- Allow complex test procedures to be constructed using a standard programming language. This means using the Visual Basic for procedure file FOR NEXT loops— and not inventing a new language.
- Have modular interconnections to other systems, such as a test results database, Division Calibration Administration and Tracking System (DCATS), and unknown future systems.
- Easily allow automatic/remote control of Exec from another program, such as manufacturing process controller, environmental test profile controller, etc.
- Keep the core simple enough to be developed in a few months. Define programmatic interfaces rich enough to allow future extensions that maintain compatibility.

Other test executives are available, but none that come close enough to the above requirements and goals.

What it is not:

These are some things you might expect in a test executive, but are not in Exec3:

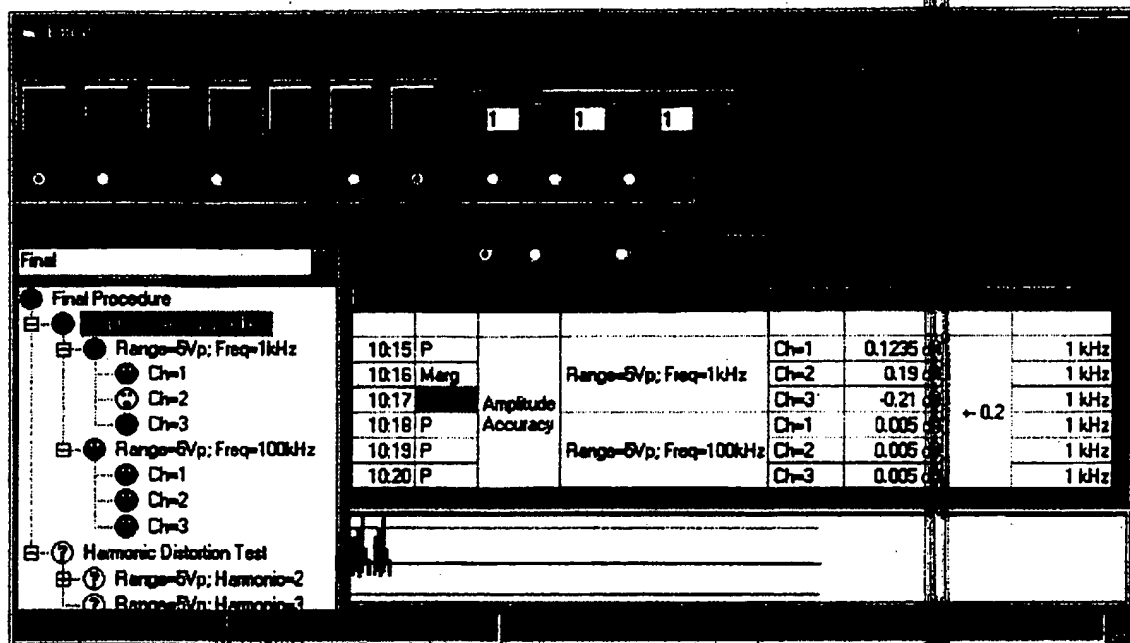
- Exec3 won't test multiple devices at once. However, automatic sequential testing of multiple devices can be accomplished by another program that controls Exec3. Also, the test software interface is designed so a future Exec could do simultaneous testing of multiple devices.
- Exec3 does not include any test equipment drivers.
- Exec3 is not compatible with Exec II test software and procedure files.
- Exec3 does not include these Exec II features: auditing/sampling plan, arrays of data, plotting of results.
- Exec3 does not include things like a test results database, equipment calibration checking, or environmental chamber control software. However, these items can interface with Exec3 through plug-ins.

Definitions

Procedure	An ordered list, sequence, or script, of <i>Tests</i> to be run. For each test, the procedure includes a list of <i>Measurements</i> and <i>Datapoints</i> , which include parameters that are passed to the test. Procedures have names like "Turn-on", "Final Sales", and "Environmental". Exec3 "runs" a procedure. Exec3 views procedures as input data, not code.
Test	A group of <i>measurements</i> in a <i>procedure</i> that share the same test algorithm, and so the same test software code. Tests have names like "Amplitude Accuracy", "Harmonic Distortion". To run a procedure, Exec3 repeatedly calls a Test for each Measurement and Datapoint. To Exec3, tests are code -- not data.
Measurement	A configuration or setup for a <i>Test</i> . Tests are <i>parameter-driven</i> , and tests get their parameters from a Measurement. Measurements have names like "Range=10V;Freq=100kHz" or "Harmonic=3". Exec3 views measurements as data to be passed to a test.
Datapoint	A sub-set of a Measurement, containing additional parameters that select a result when one measurement generates multiple results. Datapoints have names like "Channel=1" or "Peak".
Result	Datapoint Result. A single measured value, resulting from running one <i>Measurement</i> of a <i>Test</i> , and extracting a value specified by a <i>Datapoint</i> . Each Result is compared to a <i>Spec</i> and to determine pass or fail. Results are the output of running tests.
Test Software	The software that must be added to Exec3 to test a <i>ModelFamily</i> . This includes <i>Tests</i> containing test code, <i>Specs</i> , <i>Procedures</i> of <i>Measurements</i> and <i>Datapoints</i> . Test Software also encapsulates the <i>Dut</i> and the <i>Test System</i> . Test software is usually delivered in one or more DLL files.
Spec	Specification or test limits. <i>Results</i> are compared to <i>Specs</i> . Exec3 understands Specs that are numeric limits, String match, or Boolean pass/fail. Numeric limits can include up to 3 pairs of upper/lower of limit values: marginal limits, production or line limits, and customer limits.

Notice the hierarchical relationship between Procedure, Tests, Measurement, Datapoint, Datapoint Result. If you are familiar with Exec II's hierarchy, the Measurement is new. It was created to formally handle the multiple results from one physical measurement. It should simplify test code by eliminating the need for MEAS_BLOCK[] and FNSame_as_Test().

User Interface



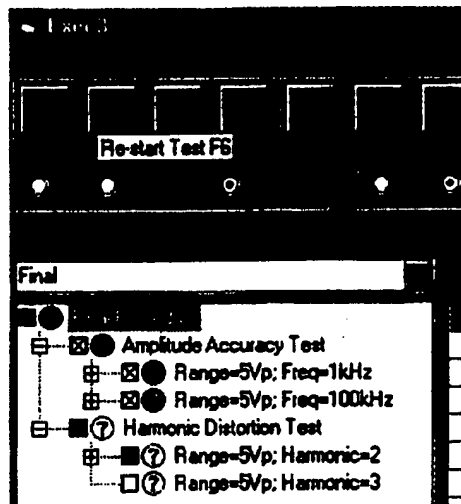
The screen from the Exec3 application is shown above. The "tape recorder" buttons near the top are used to control testing. From left to right they are Abort, re-start test, re-start measurement, pause, run, skip measurement, skip test.

The bottom of the screen is a Windows Explorer-like user interface. The procedure is shown as a hierarchy of tests, measurements, and datapoints. Colored icons indicate pass, fail, marginal, and not yet tested. The Pass/Fail icon for the procedure is a summary of all tests, with failures taking precedence. The right side shows a spreadsheet-like view of test results. They can be sorted by clicking on the top column.

The bottom right graph functions as a progress bar. Its length represents the entire procedure. The gray horizontal lines represent normalized upper and lower specification limits. Each datapoint results in a colored vertical line whose length is proportional to the ratio of value and spec, and whose color is coded for pass/fail/marginal.

User Procedures

The user can create a "User Procedure" that is a sub-set of a standard test procedure. Clicking the "Select" radio button in "Run All, Marg Fail, Select" adds checkboxes to each item in the procedure.



Dim checkboxes indicate that some of the lower-level items are checks and some are not checked. The user procedure selections can be saved to a file.

Menu Items

File

Open Test Results... opens a comma-separated-value (CSV) results file. Results are loaded into grid on the right side of the screen, and can be compared to current results, or be the basis for a re-test of "Marg/Fail" datapoints.

Open User Procedure... opens a previously saved user procedure file.

Save User Procedure... opens a previously saved user procedure file.

Print... opens a previously saved user procedure file.

ModelFamily

Displays a list of model families that can be tested. Selecting a new model family unloads the current test software and loads the new test software.

Dut

Displays a screen for entry of DUT model, serial number, options, etc. Changing the DUT clears the test results.

Settings

Displays a screen for viewing and changing Exec settings. These include: Operator, Comments, Temperature, Humidity, Qualification test, Region, Production or Customer specifications.



Tools

Add Note... allows the operator to insert a note in the test results log.

Test System Devices... Displays a spreadsheet with a row for each device in the test system. The operator can enter device information such as Maintenance Number, Service Due Date, and Address. Clicking the "Save" button will write this information to a file which will be loaded each time Exec is started.

Test System... Displays the user interface for the current test system. What appears when this is clicked depends on the currently loaded test system software.

<Plug-In 1>... Displays the user interface for the first plug-in loaded. The name of this menu item and what appears when it is clicked depends on the plug-in. There may be several plug-ins.

Test Results Files

Exec saves test results into comma-separated-value (CSV) result files. These files can be opened by Exec for review or printing. When opening a file of previous results, Exec can update the colored "pass/fail" icons in the left side of the screen. This feature makes it easy to re-run only the measurements that failed earlier (in other words, run a Marg/Fail procedure).

The CSV files can be also be opened spreadsheet program like Excel for analysis.

Exec writes measurement results to the file at the end of a test. Aborted test won't normally generate results.

File Locations and File Names

Executive stores test results files in a separate directory for each Dut model number. These Dut Model directories are located in a directory you specify in the ExecConfig.ini configuration file (see Administering Exec3). This directory can be a shared network drive.

Exec forms the results file name using the procedure name and the procedure start time/date. Exec creates a Results file subdirectory for each Serial number (or AvId).

For example, for model E1438A, Serial number "US3800123", procedure "Final", started on November 30 at 1:45:59 PM, the subdirectory and file name would be:

E1438A\US3800123\Final_1130_134559.csv

This subdirectory will be located in the directory specified by the ResultsDir= line of the ExecConfig.ini file.

File Format

See the "Exec Details" section of this document.

Writing Test Software

This section describes how you, a test software developer, will write tests for production, environmental qualification, or other special test needs. The test software developer is typically a New Product Introduction Engineer, Production Engineer, or Technician. It is assumed that you are familiar with the latest version of Microsoft Visual Basic, collections, the `implements` keyword, ActiveX components, and object-oriented programming. Test Software can be developed using other COM/ActiveX compatible languages, but that is beyond the scope of this document.

The Exec environment includes two "Wizards" that can help you write Test Software. These wizards are described at the end of this section. If you plan use the Wizards, you should read all of this section so you understand the code the wizards generate.

Overview

Before looking at example code, you should understand the Procedure-Test-Measurement-Datapoint hierarchy, the order in which test software is executed, and the files in a typical TestSw project.

Procedure Hierarchy

Every test executive, it seems, has a different approach to structuring test and naming its elements. Figure 1 diagrams the ModelFamily-Procedure-Test-Measurement-Datapoint test hierarchy used by Exec 3.

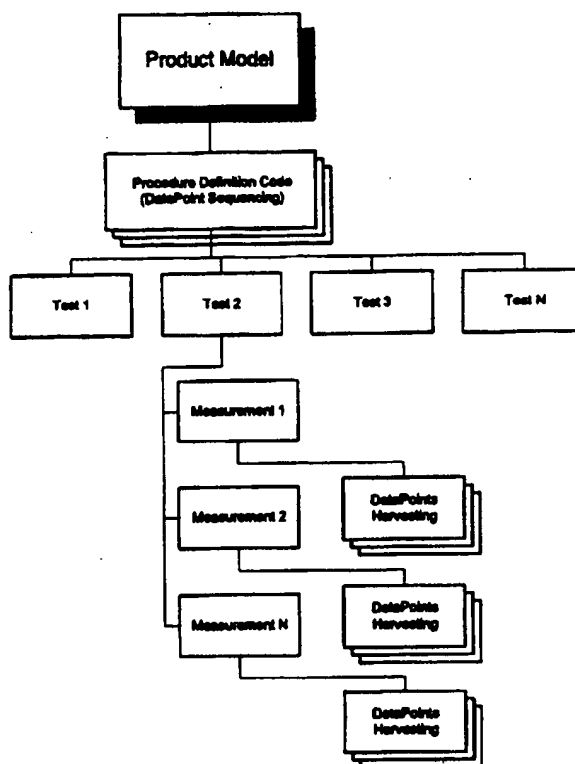


Figure 1: Test-Measurement-Datapoint Hierarchy

Components of this architecture are described next.

Product Model Family: The top level of this diagram is the Product Model. In Exec3 this level corresponds to a Test Software DLL file you create. You can develop one Test Software DLL file to test a family of specific model numbers, like E1434A/B/C, so this level is called "Model Family." Your Test Software (or TestSw) DLL contains Procedures, Tests, and so on. The entire TestSw package is typically one Visual Basic project, and can kept under revision control.

Procedure: An ordered list, sequence, or script, of *Tests* to be run. For each test, the procedure includes a list of *Measurements* and *Datapoints*. Procedures have names like "Turn-on", "Final Sales", and "Environmental". Exec3 "runs" a procedure. Your test software defines these procedures, but the test operator can select a subset of a procedure to run. Exec3 views procedures as input data, not code. A procedure is transferred from the Test Software to Exec3 as a structure of COM objects. You define a procedure by writing code to build this structure of COM objects.

Test: A group of *measurements* in a *procedure* that share the same test algorithm, and so the same test software code. Tests have names like "Amplitude Accuracy", "Harmonic Distortion". To run a procedure, Exec3 repeatedly calls a Test for each Measurement and DataPoint. To Exec3, tests are code – not data. You implement a test by adding a Visual Basic Class module to your TestSw project. The code you put in this class should implement the test algorithm in a parameter-driven way. Then, in the procedure definition code, you insert code to create an instance of this class and add it to the procedure.

Measurement (Meas): A configuration or setup for a *Test*. Each measurement within a Test can have different setup or configuration parameters. Tests are *parameter-driven*, and tests get their parameters from a Measurement. Measurements have names like "Range=10V;Freq=100kHz" or "Harmonic=3". Exec3 views measurements as data to be passed from a Procedure to a Test. You define a Measurement for a Test by creating a new *Meas* object and adding it to a Test in a Procedure. The CMeas class is already defined by Exec, so you need only to create and use Meas objects.

"Measurement" is also a phase of test execution. During the measurement phase of test execution, the measurement is started but data is not collected: this allows for multiple DUTs to be configured and triggered together.

DataPoint (Dp): A sub-set of a Measurement, containing additional parameters that select a result when one measurement generates multiple results. Datapoints have names like "Channel=1" or "Peak". One Measurement may return one or more Dps as logically makes sense to you. Some examples of multiple Dps for a measurement: minimum and maximum of a spectrum analyzer sweep, or each channel of device. If a Measurement has only one Dp, it may be blank.

"Datapoint" is also a phase of test execution. This is where the data is harvested. It doesn't make sense to separate the Measurement phase from the Dp harvesting phase, you can write your test to do the entire measurement during this phase.

Order of TestSw Execution

Figure 2 - Order of TestSw Execution, shows the order in which Exec will call the methods in your Test Software.

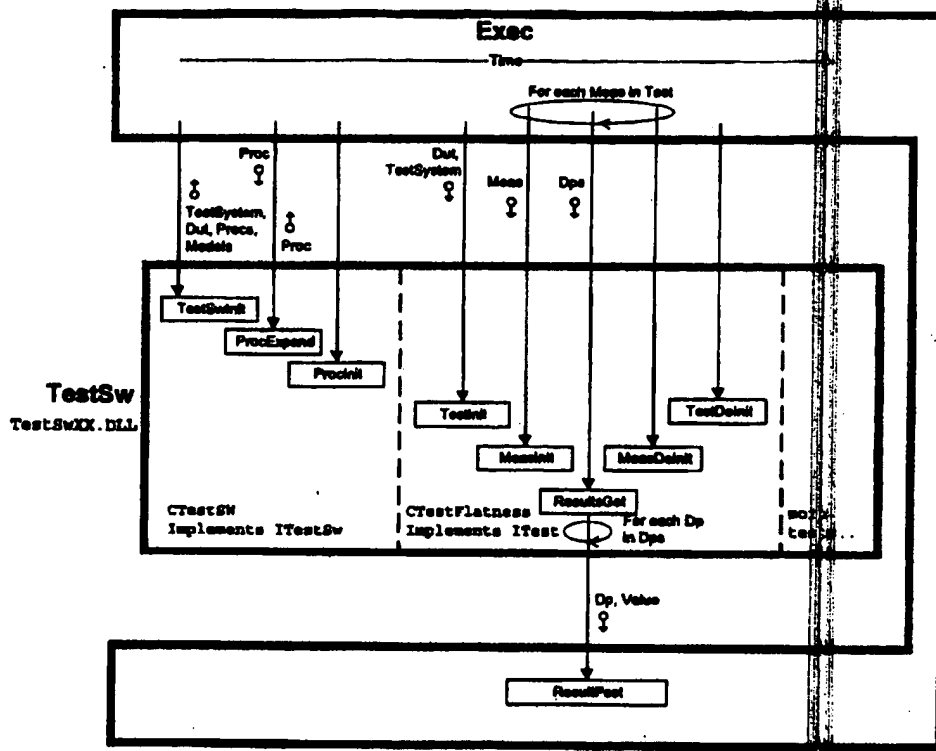


Figure 2 - Order of TestSw Execution

In this figure, the outer polygon represents Exec's code, and the middle box represents your TestSw DLL code. The left portion shows three methods you must provide in a class named "CTestSw." The middle of this box shows an example Test class, named "CTestFlatness." Usually there are more Tests, but this diagram shows only one. Each test must have the methods shown.

The methods in this figure are arranged from left to right, in the order that Exec will call them. The following describes each phase of the sequence.

1. Exec calls **TestSwInit**. This occurs immediately after Exec loads your TestSw DLL. Exec calls TestSwInit to get information from your test software, including a list of models can be tested, a list of procedure names, and references to an objects representing the DUT and test system.
2. Exec calls **ProcExpand** when the operator selects a procedure from the list of procedures. Your ProcExpand method builds the structure of Test, Meas, and Dp objects that define the details of the Procedure. When this is returned to Exec, it displays Procedure to the operator.
3. When the operator starts a procedure, Exec will call **ProcInit** in your TestSw. Your ProcInit code should initialize test system I/O, etc.

4. Exec calls the **TestInit** method in the first test. In this method, you may want to preset the test system, preset the DUT, and do setups that are common for all measurements in the test. Pass parameters for this method include a reference to the DUT object, which has properties for Address, Slot, etc.
5. Exec enters a loop to call the **MeasInit**, **ResultsGet**, and **MeasDeInit** methods of the test, once for each Measurement. The **MeasInit** pass parameters include one **Meas** object. This **Meas** object is one you created and added to the Procedure in your **ProcExpand** method. The **Meas** object can include many **Parameter** objects, which define the configuration for the measurement. So, your code in the **MeasInit** method will typically use the values of these **Measurement Parameters** to configure the Test System and DUT.
6. Exec calls **ResultsGet** in your test, and passes it a collection of **Datapoints** for the Measurements. These **Dp** objects are ones you created and added to the Procedure in your **ProcExpand** method. For each **Dp**, Exec expects your **ResultsGet** method to "call back" to Exec's **ResultPost** method. The pass parameters you send to **ResultPost** include a measured datapoint value you want Exec to compare to specification.
7. Exec calls **MeasDeInit** before calling **MeasInit** for the next Measurement.
8. When all Measurements have been run, Exec calls **TestDeInit** before moving to the next test.
9. Not shown in the figure are these **DeInit** methods: **ProcDeInit** will be called when the procedure finishes or is aborted, **ProcCollapse** will be called before changing to another procedure, and **TestSwDeInit** will be called before unloading the **TestSw**.

Test Software Project Files

As a minimum, your **TestSw** Visual Basic project will contain these class files. These VB project files are compiled into the **TestSw DLL** file. The names below are for an example **TestSw** project to test models E1234A/B. It is based on the files in the **Exec\TestSw\TestSwExample**, except "Example" is replaced with "E1234". As explained in the "Exec 3 Details" section of this document, class module names start with "C", and are saved in a file name that does not include the "C" prefix.

■ **CTestSw** (**TestSw.cls**)

This class is where the **TestSwInit**, **ProcExpand**, and **ProcInit** methods are implemented. This class is the initial interface between your **TestSw** and Exec, so the module name must be "CTestSw", and this module must implement the **ITestSw** interface defined by Exec.

■ **CDutE1234** (**DutE1234.cls**)

■ **CTestSystemE1234** (**TestSystemE1234.cls**)

These classes represent your DUT and TestSystem. Your code should use the VB **Implements** keyword so your classes are derived from the **IDut** and **ITestSystem** class defined by Exec.

■ **CTestAmplitudeAccuracy** (**TestAmplitudeAccuracy.cls**)

■ **CTestFlatness** (**TestFlatness.cls**)

These are examples of 2 tests.

■ **FPT** (**FPTMath.bas**)

This is an example of an additional file that is specific to your test software, and unrelated to Exec.

**This Page is Inserted by IFW Indexing and Scanning
Operations and is not part of the Official Record**

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images include but are not limited to the items checked:

- ☒ **BLACK BORDERS**
- ☐ **IMAGE CUT OFF AT TOP, BOTTOM OR SIDES**
- ☐ **FADED TEXT OR DRAWING**
- ☒ **BLURRED OR ILLEGIBLE TEXT OR DRAWING**
- ☐ **SKEWED/SLANTED IMAGES**
- ☒ **COLOR OR BLACK AND WHITE PHOTOGRAPHS**
- ☐ **GRAY SCALE DOCUMENTS**
- ☒ **LINES OR MARKS ON ORIGINAL DOCUMENT**
- ☐ **REFERENCE(S) OR EXHIBIT(S) SUBMITTED ARE POOR QUALITY**
- ☐ **OTHER:** _____

IMAGES ARE BEST AVAILABLE COPY.

As rescanning these documents will not correct the image problems checked, please do not report these problems to the IFW Image Problem Mailbox.